Assignment 2
DCS3101

Kent Odde

October 7, 2020

Contents

Abstract

This is the submission for the second assignment in the course DCS-3101, Introduction to Cybersecurity, at USN Kongsberg, fall of 2020.

Q1

There are several ways of displaying an algorithm, I have chosen to include a visual representation first:

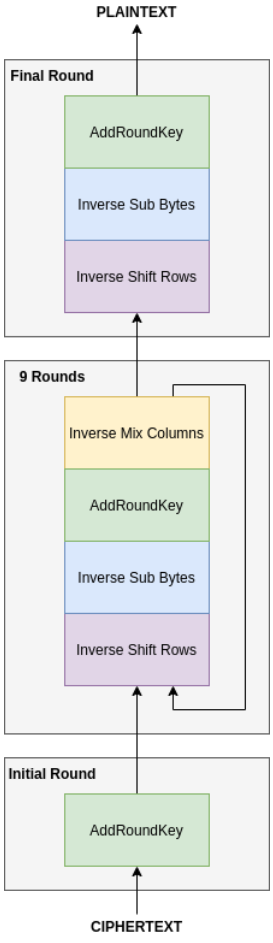

Figure 1: Algorithm for decrypting AES

However, as code is perhaps the most precise way to describe an algorithm, and I already happen to have implemented AES encryption in C++, i decided to also include the decrypt function from that implementation:

```
std::string AES::decrypt128BitMessage(std::string cipherText,
    std::string keyString)
{
   auto key = generateGridFromHexString(keyString);
   auto grid = generateGridFromHexString(cipherText);
   auto expandedKey = generateExpandedKey(key);

   //Initial Round:
   addRoundKey(grid, expandedKey.at(expandedKey.size() - 1));

   //9 Rounds
   for(int i = expandedKey.size() - 2; i > 0; i--)
   {
      invShiftGrid(grid);
      invSubstitute(grid);
      addRoundKey(grid, expandedKey.at(i));
      invMixColumns(grid);
   }

   //Final Round
   invShiftGrid(grid);
   invSubstitute(grid);
   addRoundKey(grid, expandedKey.at(0));

   return gridToHexString(grid);
}
```

Q2

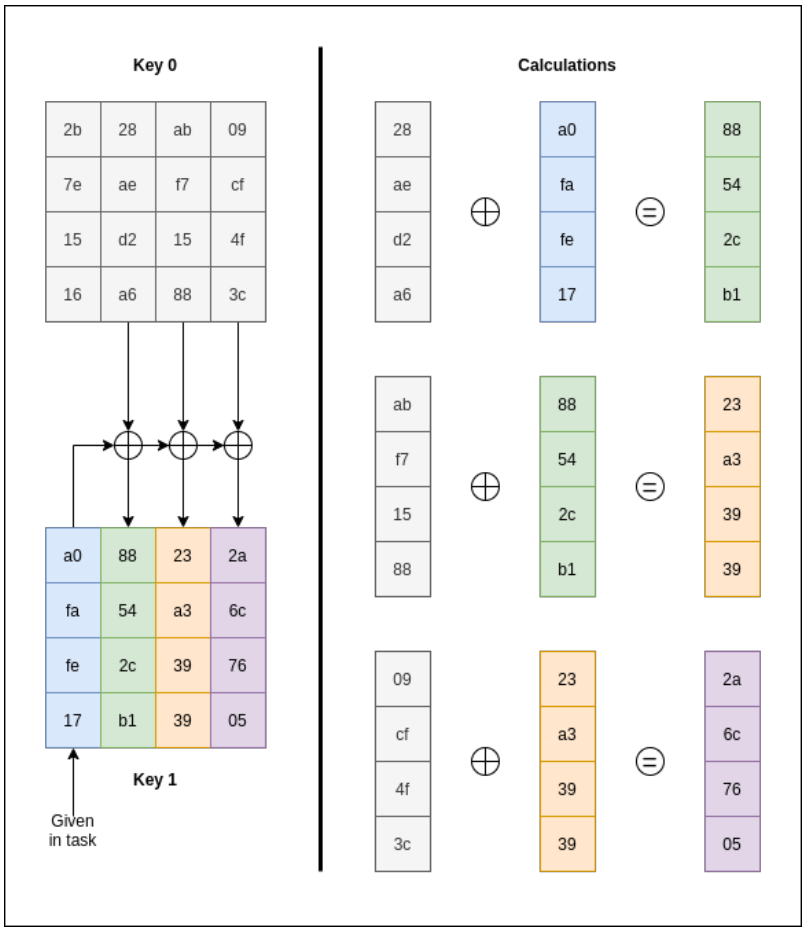

Figure 2: Calculating the first round key

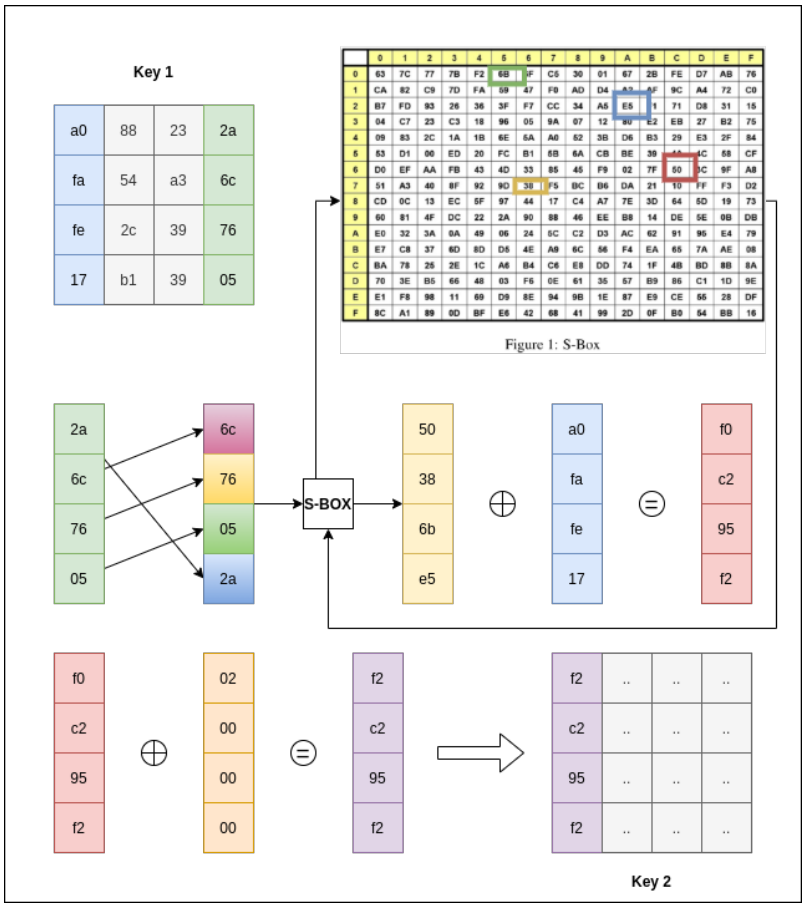

Figure 3: Calculating the first column of the second round key

As mentioned I have already implemented the AES algorithm, so have chosen to include the full expanded key generated by my program:

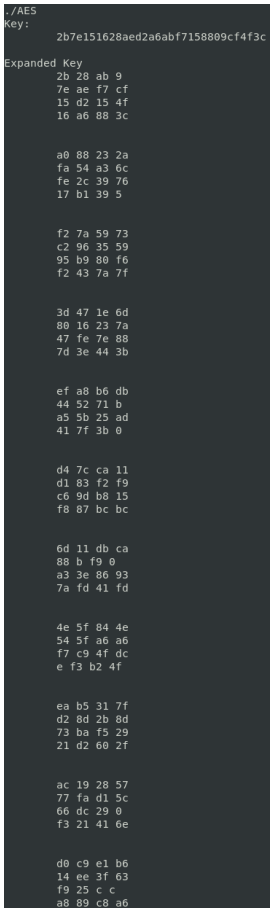

Figure 4: Expanded key

Q3

ECB, CBC, OFB etc. are modes of operation in stream ciphers or when it comes to encrypting data that is longer than the block size given within a certain block cipher algorithm. For instance in AES, a block is 128 bits. If you however want to encrypt more data than this, we use for instance CBC in order to provide additional security across the blocks. The principle is to treat the blocks as a stream.

In this task we have been given these parameters for demonstrating the different modes of operation:

- Plaintext: *DCS-3101*
- IV: *NO*
- Key: *EU*
- Block size: *16 bits*
- Encryption: *XOR*

The block size of sixteen bits means that each block will have two characters, and we will end up with four blocks.

Converted to binary, the plaintext string looks like this:

01000100 01000011 01010011 00101101
00110011 00110001 00110000 00110001

The block size of sixteen bits means that each block will have two characters, and we will end up with four blocks.

DC	:	01000100 01000011
S-	:	01010011 00101101
31	:	00110011 00110001
01	:	00110000 00110001

The key and IV converted to binary looks like this:

IV	:	01001110 01001111
KEY	:	01000101 01010101

1. ECB

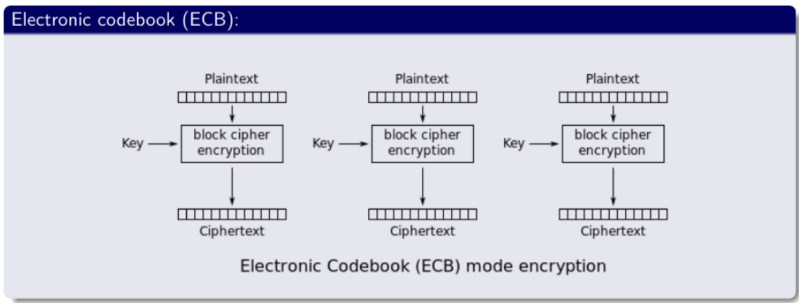

Figure 5: ECB encryption diagram from lecture

ECB is the simplest of the three modes of operation discussed in this assignment, and provide no additional security. The principle is to divide the text into blocks, encrypt them with whatever way we like, and concatenate the ciphertexts produced. This means that if you break one of the blocks, you will be able to break them all. However, an error will not propagate through the blocks.

In practice it will look like this:

```
01000100 01000011   01010011 00101101   00110011 00110001   00110000 00110001
          XOR                 XOR                 XOR                 XOR
01000101 01010101   01000101 01010101   01000101 01010101   01000101 01010101
                                    =
00000001 00010110   00010110 01111000   01110110 01100100   01110101 01100100
```

2. CBC

As the complexity grows in the following operation modes, we will from here on use hexadecimal numbers for increased readability.

In hex, the input data, looks like this:

DC	:	44 43
S-	:	53 2d
31	:	33 31
01	:	30 31

The key and IV converted to binary looks like this:

IV	:	4e 4f
KEY	:	45 55

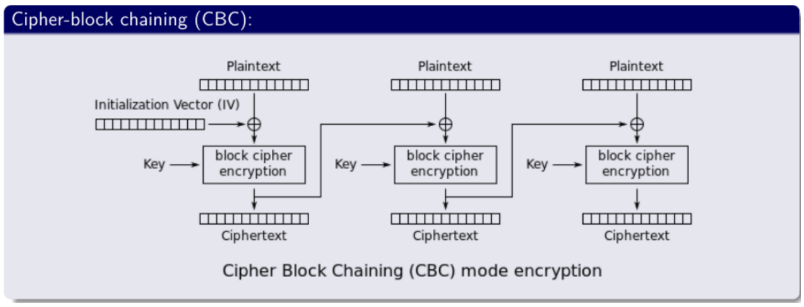

Figure 6: CBC encryption diagram from lecture

CBC stands for cipher-block chaining, and that is exactly what it does. Each plaintext block is XORed with the ciphertext of the previous block, creating a chain. For the first block, we XOR it with an IV (initialization vector). This ensures that each block is dependent on the previous one, and increases the security, as the breaking of one block, does not break the entire message. It does have error propagation however, so an error in a block, wil give an error in all the later blocks.

In practice it looks like this:

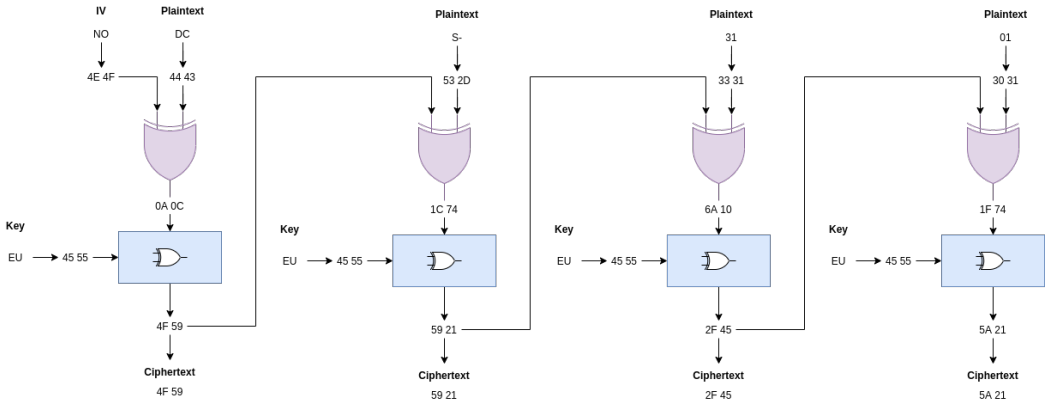

Figure 7: Illustration of CBC

So the encrypted message becomes:

CBC Encrypted: 4F 59 59 21 2F 45 5A 21

3. OFB

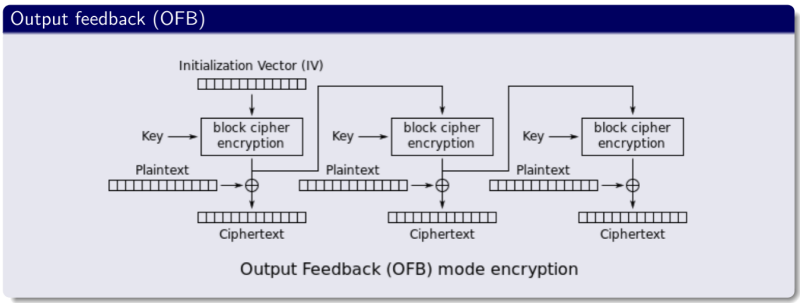

Figure 8: OFB encryption diagram from lecture

OFB (Output feedback), does the block encryption on the IV, rather than on the plaintext. It then XORS the plaintext with the output of the block encryption. The input in the following block encryptions however is the output of the previous block cipher (which does not involve the plaintext). Because of this no blocks are dependent on the plaintext or ciphertext of the previous block, which means that an error will only give an error in the current block (as we will see in the next part).

In practice it looks like this:

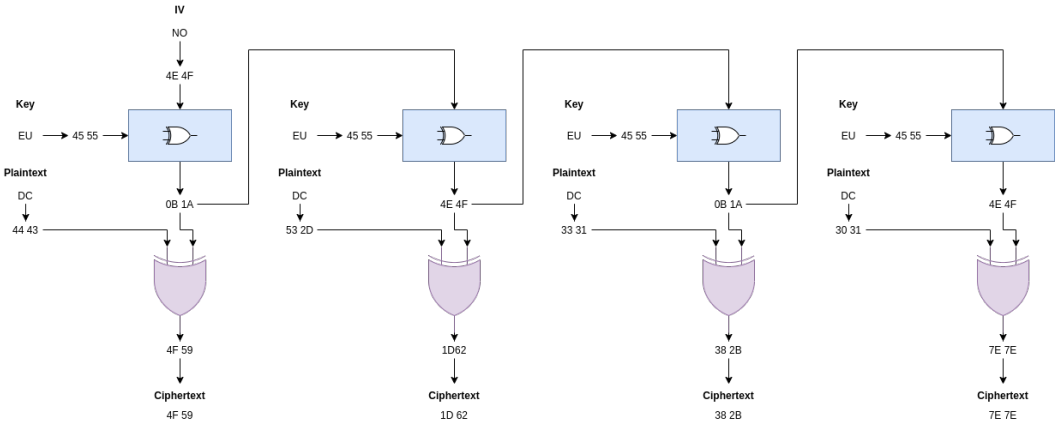

Figure 9: Illustration of OFB

OFB Encrypted: 4F 59 1D 62 38 2B 7E 7E

4. OFB - Decryption

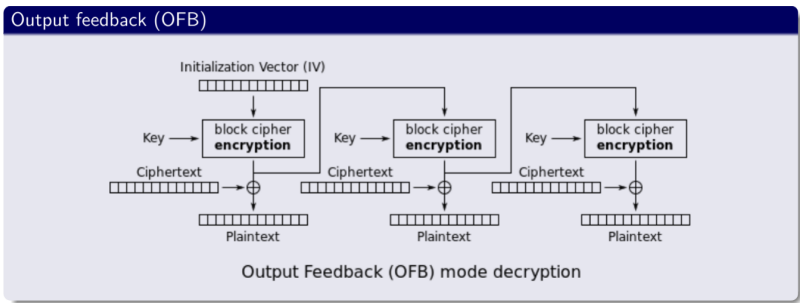

Figure 10: OFB decryption diagram from lecture

In this assignment we will flip the last bit of the first output of the first block from the previous task, and see the effect it has on the decrypted message.

The ciphertext with the flipped bit, looks like this:

Ciphertext: 4F 58 1D 62 38 2B 7E 7E

The fourth hex digit 9, turned into an 8.

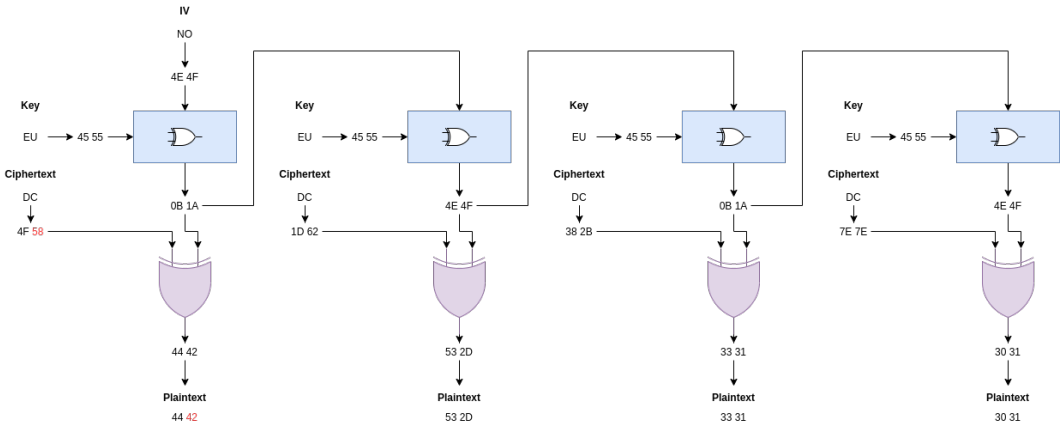

Figure 11: Illustration of OFB decryption with a flipped bit

Decrypted message:

```
Plaintext:      44 42 53 2D 33 31 30 31
```

If we take the decrypted message and turn it back into text, it will look like this:

```
Plaintext:      DBS-3101
```

We can now see that the original C, has turned into a B. This corresponds with what we mentioned in the previous task, that the bit error only has an effect on one block.

Comparision

If we compare the results from the three modes:

```
ECB:        01 16 16 78 76 64 75 64
CBC:        4F 59 59 21 2F 45 5A 21
OFB:        4F 59 1D 62 38 2B 7E 7E
```

We of course see that they are drastically different. ECB is without the IV, so it will of course bear the least resemblance to the others. We can also see that while CBC and OFB start of similarly, they quickly diverge.

Q4

In cryptography it is common to separate between two different types of encryption, namely *symmetric* and *asymmetric*.

Symmetric encryption is the intuitive and oldest way of encrypting data, where one uses the same key to both encrypt and decrypt the data. There are countless examples of this, where perhaps the most common one today is AES. However the Caesar Cipher, Vignere Cipher and all encryption we did in task 3 are also examples of symmetric encryption.

Asymmetric encryption (also known as public key cryptography), differs from this, as it uses different keys for encryption and decryption. If Alice encrypts a message, Bob will not be able to decrypt the message with the same key. He will have to have the sibling key in the key-pair, to be able to make sense of the encrypted message. The most common example of asymmetric encryption is RSA.

In symmetric cryptography, we can usually in principle select any key we like. There are no inherent properties necessary within the key, so key generation is quite trivial. In asymmetric cryptography however, this is not the case. To generate an asymmetric key-pair, we use mathematics to generate them.

In the case of RSA, we use modulo arithmetic and large primes, and take advantage of the fact that it is difficult to factor two large primes multiplied together.

The term public key cryptography comes with an interesting possibility that opens up when you have asymmetric encryption. Usually one takes the keys generated and refer to one of them as the public key, and the other as the private key. The public key you share openly while keeping the private key secret.

The reason for doing this, is to let anyone encrypt data with the public key, whilst knowing that you will be the only one able to decrypt it. This is quite practical in and of itself, but in addition to this, if you encrypt data with your private key, anyone who unlocks it with your public key can be sure that the integrity of the message has not been compromised.

These properties are not present in symmetric cryptography. Because of this, and the fact that the keys are generated with a lot more care, one usually argues that asymmetric cryptography is safer than symmetric cryptography.

Still, symmetric cryptography has its place, as the complexity of asymmetric cryptography is much higher and requires more overhead.

As I have understood it, a common approach is to use asymmetric cryptography,

in order to agree on a symmetric key, and then do large data-transfers in the symmetric way.

Appendices